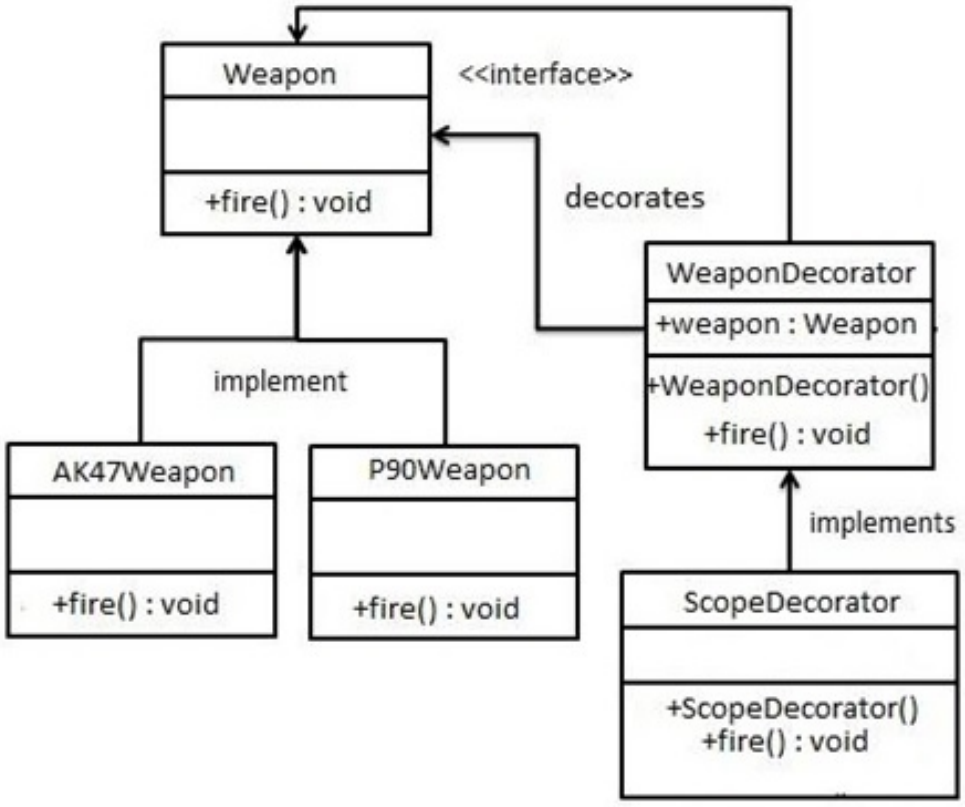
Breviar laborator 14

1. Design Patterns

Decorator

De multe ori in cadrul unei aplicatii apare urmatorul scenariu. Aveti de implementat o anumita functionalitate care trebuie sa poate fi extinsa fara sa stiti in momentul proiectarii in ce mod va fi extinsa. De exemplu lucrati la un joc de tipul shooter. Aveti o lista de arme pe care un user le poate achizitiona si utiliza pentru a-si impusca oponentii. La un moment dat observati ca incasarile din joc incep sa scada o data ce alternativele de pe piata ofera mai multe feature-uri pentru armele lor. Pentru a nu da faliment va trebui sa adaugati niste feature-uri noi armelor voastre, dar trebui sa tineti cont ca orice modificare de cod creste fragilitatea si sansa de aparitie a unor bug-uri care initial nu existau. Rezolvarea este foarte simpla: O sa creati o clasa abstracta wrapper peste o instanta de arma. Daca de exemplu vreti ca toate armele voastra sa poata fi mortale dar silentioase, creati clasa WeaponSilencer ce implementeaza interfata de baza a armelor si are un obiect de tip arma intern. In metoda fire, in decorator, o sa setati eventual sunetul jocului mai incet.



Un alt exemplu la indemana sunt fluxurile din java.

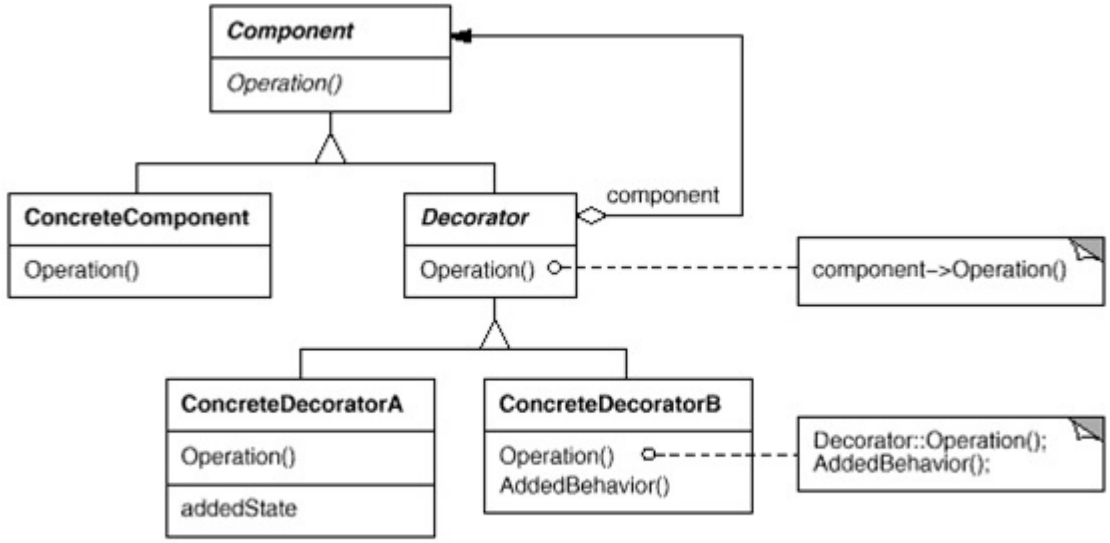
```
CipherOutputStream cos = new CipherOutputStream(new FileOutputStream( “file” ) , c
);

PrintWriter pw = new PrintWriter (new OutputStreamWriter ( cos ) );

pw.println ( “Stand and unfold yourself” );
```

```
pw.close( );
```

La baza avem un flux de iesire care va scrie in fisierul "file" niste octeti. CipherOutputStream este o clasa decorator si adauga peste fluxul de octeti proprietatea ca acesta este criptat. Deoarece aceasta clasa nu ofera suport de lucru decat pentru octeti avem nevoie de un alt decorator care sa ne permita sa adaugam text in fluxul de iesire. De aceea folosim clasa PrintWriter. Observati ca atat PrintWriter, OutputStreamWriter cat si CipherOutputStream sunt doar niste decoratoare ele modificand intr-un anumit mod fluxul, totusi acest flux trebuie sa apara de undeva si de accea toata aceasta decorare trebuie sa aiba la baza un flux instantiat (FileOutputStream).



Mapand pe diagrama exemplul nostru, FileOutputStream joaca rolul de componenta concreta, PrintWriter, OutputStreamWriter si CipherOutputStream joaca rolul de decoratoare concrete.

Atentie: Daca in tot lantul de decorari nu exista un decorator care sa aiba la baza o componenta concreta constructia nu va functiona.

2. Fluxuri

Dennis Ritchie implementeaza in 1984 primul sistem I/O pe baza de stream in cadrul sistemului de operare Unix. Acest concept are la baza crearea unui canal de comunicatie intre doua entitati: sursa si destinatie. Sursa scrie informatii in canalul de comunicatie, iar destinatia poate sa citeasca aceste date, canalul permitand trecerea fluxului de date intr-o singura directie.

Fluxurile pot fi clasificate dupa directia canalului de comunicatie:

- fluxuri de intrare(pentru citirea datelor),
- fluxuri de iesire (pentru scrierea datelor);

dupa tipul de date pe care le opereaza:

- fluxuri de octeti (transfer serial pe 8 biti),
- fluxuri de caractere (transfer serial pe 16 biti);

sau dupa actiunea lor:

- fluxuri primare de citire/scriere a datelor,
- fluxuri pentru procesarea datelor.

Datorita faptului ca exista doua directii de comunicare, exista doua tipuri mari de stream-uri pentru orice nod de comunicatie: input stream si output stream. Tastatura ar fi un exemplu de input stream, iar monitorul un output stream. Sursa si destinatia nu trebuie sa fie neaparat periferice, ele pot fi si module soft.

Figure 1: input stream

Figure 2: output stream

Fluxuri de octeti

Programele folosesc fluxuri de octeti pentru a citi sau scrie date pe 8 biti (un octet). Pentru o mai buna intelegere a acestor clase, se va propune spre analiza implementarea clasei Copy care contine metode care asigura copierea unui fisier text in alt fisier text.

```
public class Copy {
  private String input , output ;
  public Copy( String input , String output ) {
      this.input = input ;
      this.output = output ;
  }
  public void copyFile () {
      InputStream in =null;
      OutputStream out =null;
      try {
          in =newFileInputStream ( input ) ;
          out =newFileOutputStream ( output ) ;
          while( in .available() > 0)
              out.write(in.read ()) ;
      }
      catch( FileNotFoundException e ) {
          e . printStackTrace () ;}
      catch( IOException e ) {
          e . printStackTrace () ;}
      finally {
          try{
              if( out !=null) out.close();
              if( in !=null)  in.close() ;
          }
          catch( IOException e ) {
              e . printStackTrace () ;}
      }
  }
}
```

Fluxurile la nivel de octet utilizeaza doua ierarhii avand drept clase radacina: InputStream si OutputStream. Exista mai multe clase pentru manipularea fluxurilor de octeti, folosite in aceeasi maniera, avand moduri diferite de contructie. Cele mai utilizate clase sunt FileInputStream si FileOutputStream.

IMPORTANT! Fisierele text trebuie sa se afle in folderul radacina al proiectului care contine clasele care prelucreaza informatia din aceste fisiere.

Fluxurile de caractere

Limbajul Java stocheaza valorile de tip caracter folosind conventii Unicode. Fluxurile de caractere I/O isi translateaza automat formatul intern de la setul de caractere locale. Toate clasele de fluxuri de caractere mostenesc clasele Reader si Writer. Ca si in cazul fluxurilor de octeti, exista clase de fluxuri de caractere specializate pentru operatiile I/O cu fisiere: FileReader si FileWriter. Se va oferi un exemplu elocvent pentru a intelege mai bine uzitarea acestor clase.

```
public class Read {
  private String input;
  public Read( String input ) {
      this.input = input ;
  }
  public void read () {
      FileReader stream = null;
```

```
      BufferedReader br = null;
      try{
          stream = new FileReader ( input ) ;
          br = new BufferedReader ( stream ) ;
          String line = br.readLine () ;
          while (line != null) {
              System .out .println (line) ;
              line= br.readLine () ;
          }
      }
      catch(FileNotFoundException e ) {
          e . printStackTrace () ;}
      catch ( IOException e ) {
          e . printStackTrace () ;}
      finally {
          try {
              if( stream !=null) {
                  stream.close() ;}
              if( br !=null) {
                  br.close() ;}
          }
          catch( IOException e ) {
              e . printStackTrace () ;}
      }
  }
  public static void main ( String args [ ] ) {
      Read r = new Read("fisier1.in") ;
      r.read () ;
  }
}
```

Observatie: Obiectul BufferedReader creaza un flux de intrare a caracterelor cu stocare temporara (si posibilitate de citire a caracterelor sub forma de linii) din fluxul de intrare a caracterelor primit ca parametru (stream), utilizand dimensiunea implicita a tabloului intern.

Fluxuri cu zone tampon

Pentru un flux I/O fara zone tampon fiecare cerere de citire sau scriere este administrata direct de sistemui de operare. Aceasta face ca programul sa fie mai putin eficient, deoarece fiecare cerere declanseaza accesul la disc, activitate in retea sau alte operatii care consuma timp. Pentru a reduce timpul de procesare a fluxurilor, platforma Java implementeaza fluxuri I/O cu zone tampon. Fuxurile de intrare cu zone tampon citesc datele dintr-o zona de memorie cunoscuta ca tampon (buffer); API –ul de intrare este apelat numai cand tamponul este gol. Similar, fluxurile de iesire scriu datele in zona de tampon si API-ul de iesire este apelat atunci cand tamponul este plin. Un program poate converti un flux fara zona tampon intr-un flux cu zona tampon astfel: un obiect de tip flux fara zona tampon este trecut ca argument unui contructor pentru o clasa de tip flux cu zona tampon.

Fluxuri standard

Limbajul Java pune la dispozitia utilizatorului trei fluxuri standard pentru comunicare cu consola:

1. fluxul standard de intrare (Standard Input) - folosit pentru citirea datelor;
2. fluxul standard de iesire (Standard Output) - folosit pentru afisarea datelor;
3. fluxul standard de eroare (Standard Error) - folosit pentru afisarea erorilor.

In consecinta, clasa System din pachetul java.lang contine trei referinte statice de urmatoarele tipuri:

1. InputStream in;
2. PrintWriter out;
3. PrintWriter err.

Astfel, pentru Standard Input exista fluxul System.in, pentru Standard Output se uziteaza System.out, iar pentru Standard Error este System.err.

Observatie: Nu este necesara instantierea acestor trei stream-uri, deoarece ele se deschid automat inaintea executiei aplicatiei. Celelalte stream-uri se deschid in momentul crearii lor, prin apelul constructorului clasei corespunzatoare. De asemenea, acestea nici nu inchid prin apelul metodei close().

Pentru inceput, se va prezenta un exemplu in care se uziteaza obiecte de tip InputStreamReader, BufferedReader si o instanta statica din clasaSystem. Clasa Suma contine o metoda care citeste de la tastatura doua numere intregi, iar, pentru simplitate, acestea sunt citite ca obiecte de tip String, urmand a fi convertite.

```
public class Suma {
  int a, b ;
  public void read () {
      InputStreamReader stream= null;
      BufferedReader br= null;
      try{
          stream = new InputStreamReader ( System.in ) ;
          br = new BufferedReader ( stream ) ;
          String line = br.readLine () ;
          a = Integer.parseInt ( line ) ;
          line = br.readLine () ;
          b = Integer.parseInt ( line ) ;
      }
      catch( FileNotFoundException e ) {
          e.printStackTrace () ;}
      catch ( IOException e ) {
          e . printStackTrace () ;}
      finally{
          try{
              if( stream !=null)
                  stream.close() ;
              if ( br != null)
                  br.close() ;
          }
          catch ( IOException e ) {
              e . printStackTrace () ;}
      }
  }
}
```

In al doilea exemplu, se va prezenta citirea de la tastatura a doua numere intregi,utilizand un obiect de tip Scanner. Aceasta clasa pune la dispozitie o serie de metode pentru citirea celor mai importante tipuri predefinite de date din limbajul Java.

```
public class Test {
  public static void main( String args[ ]) throws Exception {
      Scanner in = new Scanner ( System . in ) ;
      int a, b ;
      a = in . nextInt () ;
      b = in . nextInt () ;
      int result;
      result = a + b ;
      System.out.println ( a + " + " + b + " = " + result) ;
      in.close() ;
  }
}
```